COMSC-210 Lecture Topic 9
Associative Arrays

☐ Reference
Childs Ch. 10

☐ Associative Arrays
generalization of the "key"
now may be ANY value, not just a numeric "index"
e.g., phone book

```
  phoneBook["RBurns"] = 22483;
```

other languages support associative arrays

```
  $phoneBook["RBurns"] = 22483; // PHP
  phoneBook["RBurns"] = 22483 // Python
  phoneBook["RBurns"] = 22483 // JavaScript
  phoneBook.put("RBurns", 22483); // Java
```

a.k.a. "dictionaries" or look-up tables

☐ A New `operator[]`
parameter is now *templated* instead of `int`
  old: `int key`
  new: `U key`

```
T& operator[](const U& key) // O(n)
{
  search "in use" array elements or list nodes for key match
  if not found, insert it
  return mutable reference
}

T operator[](const U& key) const
{
  search "in use" array elements or list nodes for key match
  if not found, return dummy
  return mutable reference to value
}
```

"key" can even be an int: would work like indexed array!

☐ Linked Structure Implementation

```
template <class T, class U>
class Array
{
  struct Node
  {
    T value;
    U key;
    bool inUse;
    Node* next;
  };
  ...
```

☐ Arrayed Implementation

```
template <class T, class U>
class AssociativeArray
{
  T* value;
  U* key;
  bool* inUse;
  ...
```

☐ Other Function Modifications
size: for big oh considerations, track instead of count
clear, size, capacity: no changes
containsKey, deleteKey: new parameter, no range-checking
`vector<U> keys() const;` // all "in use" keys

☐ `operator[]` Setter, O(n)
// dynamic array version

```
template <class T, class U>
T& AssociativeArray<T, U>::operator[](const U& parameter)
{
  for-loop to scan all data
    if inUse[i] == true AND key[i] matches parameter
      return mutable reference to value[i]
  for-loop to scan all data
    if inUse[i] == false
      set key[i] to parameter
      set inUse[i] to true
      add 1 to size
      return value[i]
  set i = cap
  double the array capacities
  set key[i] to parameter
  set inUse[i] to true
  add 1 to size
  return value[i]
}
```

// linked structure version

```
template <class T, class U>
T& AssociativeArray<T, U>::operator[](const U& parameter)
{
  for-loop to scan all data
    if p->inUse == true AND p->key matches parameter
      return mutable reference to p->value
  for-loop to scan all data
    if p->inUse == false
      set p->key to parameter
      set p->inUse to true
      add 1 to size
      return p->value
  add new node to end
  set end->key to parameter
  set end->inUse to true
  add 1 to size
  return end->value
}
```

☐ `operator[]` Getter, O(n)
// dynamic array version

```
template <class T, class U>
T AssociativeArray<T, U>::operator[](const U& parameter) const
{
  for-loop to scan all data
    if inUse[i] == true AND key[i] matches parameter
      return value[i]
  return dummy
}
```

// linked structure version

```
template <class T, class U>
T AssociativeArray<T, U>::operator[](const U& parameter) const
{
  for-loop to scan all data
    if p->inUse == true AND p->key matches parameter
      return p->value
  return dummy
  AssociativeArray<int, string> phoneBook;
  phoneBook["RBurns"] = 22483;
  ...
  vector<string> keys = phoneBook.keys();
  for (int i = 0; i < keys.size(); i++)
    cout << "phoneBook[" << keys[i] << "] = "
      << phoneBook[keys[i]] << endl;
```

```
}
```